НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри
_____ О.В.Коваль
(підпис) (ініціали, прізвище)
“___”_____2019 р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки
6.050103 “Програмна інженерія’’

на тему: Розробка програмних засобів для авторизованого доступу до кафедрального хмарного сховища

Виконав: студент 4 курсу, групи ТВ-51

Ковтун Євген Валерійович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник старший викладач, Ляшенко Максим Володимирович _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент доцент, Баранюк Олександр Володимирович _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки 6.050103 “Програмна інженерія’’

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2018 р.

ЗАВДАННЯ
на дипломну роботу студенту
Ковтуну Євгену Валерійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи “Створення користувацького інтерфейсу для доступу до кафедрального хмарного сховища”

керівник роботи старший викладач, Ляшенко Максим Володимирович
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”___”_____ 201__ р. № _____ __

2. Строк подання студентом роботи ___ _____ 201__ р.
3. Вихідні дані до роботи персональний комп’ютер під керуванням операційної системи Ubuntu, мова програмування Java, браузер та Інтернет.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) проаналізувати існуючі програмні рішення щодо організації хмарного сховища та користувацького інтерфейсу, обґрунтувати обрані програмні рішення та технології, розробити програмне забезпечення, що виконує дані завдання та зробити висновки за результатами роботи.
5. Перелік ілюстраційного матеріалу (з точним зазначенням обов’язкових креслень)
Актуальність, Мета та завдання, Основні модулі функціоналу системи, Опис функціональності, Групи, Управління файлами, Управління користувачами, Робота з группами, Робота з файлами, Програмні засоби що використовуються, Single Page application design, Висновки
Дата видачі завдання ”___”_____ 201__ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Вивчення та аналіз задачі	20.03.19 - 04.04.19	
2.	Розробка архітектури та загальної структури системи	04.04.19 - 15.04.19	
3.	Розробка структур окремих підсистем	15.04.19 - 25.04.19	
4.	Підготовка матеріалів	25.04.19 - 06.05.19	
5.	Програмна реалізація системи	06.05.19 - 16.05.19	
6.	Захист програмного продукту	17.05.19	
7.	Оформлення пояснювальної записки	17.05.19 - 30.05.19	
8.	Передзахист	31.05.19	
9.	Захист		

Студент _____ Ковтун Є.В
(підпис) (прізвище та ініціали)

Керівник роботи _____ Ляшенко М.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Метою роботи було створення користувацького інтерфейсу для доступу до кафедрального хмарного сховища. Система дозволяє зберігати файли та папки, а також надає методи для управління доступом до них. Система надає можливість надати доступ як окремим користувачам, так і групам користувачів, містить засоби для створення та управління групами користувачів.

Записка містить 76 сторінки 15 рисунків та 6 посилань

ABSTRACT

The purpose of this work was to create a user interface for access to institute department cloud storage. The system allows you to store files and folders and provides methods for managing access to them, the system provides the ability to provide access to both individual users and user groups, and includes tools for creating and managing user groups.

The note contains 76 pages 17 drawings and 6 links

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API (англ. Application Programming Interface) — прикладний програмний інтерфейс.

СУБД — система управління базами даних.

POJO (англ. Plain Old Java Object) — простий Java об’єкт.

EJB (англ. Enterprise Java Bean) — специфікація технології написання і підтримки серверних компонентів, що містять бізнес-логіку. Є частиною Java EE.

IoC (англ. Inversion of Control) — інверсія контролю.

ВСТУП

На сьогоднішній день кафедрам, як і будь-яким іншим організаціям необхідно мати спосіб зберігання даних, наразі найкращим з відомих способом є хмарне сховище. Дана технологія надає можливість доступу до системи, з будь якого комп'ютера, що має підключення до мережі інтернет. Вагомою перевагою хмарного сховища над іншими рішеннями є те, що вся відповідальність за захист та управління фізичним середовищем лежить на постачальнику послуг по оренді хмарного середовища. А отже кафедрі не потрібно купувати та підтримувати фізичні сервери. Теж саме стосується масштабування системи, як горизонтального (добавлення фізичних машин в пул ресурсів), так і горизонтального (збільшення ресурсів, таких як потужність процесорів, оперативна пам’ять, і т.д, в межах однієї машини), відповідальність за це також лежить на постачальнику послуг. Також це надає можливість для організації організувати спільну роботу з даними.

Для зручної роботи необхідно надати користувацький інтерфейс. Основними вимогою до інтерфейсу є те, що він повинен бути інтуїтивно зрозумілим, та з точки зору користувача має мати якомога нижчий час навчання, та звикання до нього.

Виходячи з цього метою роботи є розробка користувацького інтерфейсу для системи зберігання даних в віддаленому хмарному сховищі, надання можливості спільного доступу до даних та роботи з ними.

Розроблений додаток надає користувачам можливість завантажувати файли, створювати папки, керувати доступом до ресурсів, в тому числі і доступом для груп користувачів. Інтерфейс було розроблено враховуючи специфіку системи, а також переваги та недоліки користувацьких інтерфейсів для існуючих систем зберігання даних.

Основною цільовою аудиторією програми є співробітники та студенти кафедри, яким потрібен засіб для зручного зберігання файлів та директорій, а також засоби контролю та обмеження доступу до них.

Записка містить 5 розділів.

У першому розділі описується постановка задачі створення користувацького

інтерфейсу для доступу до кафедрального хмарного сховища.

У другому розділі наводиться огляд існуючих рішень інтефрейсів для систем зберігання даних.

У третьому розділі наведено опис засобів реалізації програмної системи.

В четвертому розділі наведено опис програмної реалізації даної системи.

У п’ятому розділі описано реалізований програмний продукт, та наведено методику роботи користувача з ним.

1. ПОСТАНОВКА ЗАДАЧІ СТВОРЕННЯ КОРИСТУВАЦЬКОГО ІНТЕРФЕЙСУ ДЛЯ ДОСТУПУ ДО КАФЕДРАЛЬНОГО ХМАРНОГО СХОВИЩА

Метою цієї роботи є створення інтерфейсу для кафедральної системи зберігання даних у хмарному середовищі. Надання інтерфейсних засобів для роботи з цими даними, створення засобів для організації та обмеження доступу до цих даних.

На данный момент одним з найвідоміших серед існуючих програмних рішень є Гугл Диск (Google Drive) від компанії Google, що було створено 24 червня 2012 року і підтримується дотепер. Основними недоліками Google Drive є відсутність зручного способу створення груп та управління ними, неможливість перегляду групп, до яких користувач належить. Також неможливо задати користувачеві для файлу чи папки рівень доступу нижче ніж у групи до якої цей користувач належить, якщо доступ до цієї папки чи файлу було надано для группи.

Також в системі Google drive кожен хто має доступ до файлу має можливість надати доступ до цього файлу іншим користувачам системи, що не завжди припустимо з причин безпеки даних.

На основі аналізу цих недоліків було сформовано вимоги до створюваної системи. Система має забезпечувати наступний функціонал:

— Завантаження файлів на сервер;

— Скачування файлів;

— Створення папок;

— Видалення папок;

— Видалення файлів;

— Надання доступу до файлів та папок окремим користувачам;

— Створення груп користувачів;

— Надання доступу до файлів та папок групам користувачів;

— Редагування рівнів доступу до окремих файлів чи папок;

— Запрошення користувачів в систему;

— Редагування прав користувача в системі;

— Видалення користувача;

Для того щоб почати використовувати систему користувач має отримати запрошення в систему, після чого перейти за посиланням, Що вказано в запрошенні, та пойти процедуру авторизації. Авторизація має відбуватися за допомогою Google через OAuth2.

Інтерфейс повинен бути написаний з використанням наступних засобів:

— мова розмітки HTML;

— мова стилів CSS;

— мови програмування TypeScript та JavaScript;

— фреймворку Angular.

Серверна частина частина має бути створена на мові програмування Java з використанням фреймворку Spring.

Результатом розробки має стати единий файл з розширенням “.war”

Також, для того щоб систему можно було використовувати необхідно розгорнути її в хмарному середовищі Heroku.

2. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ІНТЕРФЕЙСІВ ДЛЯ СИСТЕМ ЗБЕРІГАННЯ ДАНИХ

Користувацький інтерфейс є невід’ємною частиною будь якої системи. Сьогодні при створенні будь якої системи все більше і більше уваги приділяється саме користувацькому інтерфейсу, адже саме він визначає наскільки система буде привабливою з точки зору користувача, і наскільки активно цю систему будуть використовувати.

В даному розділі було проведено аналіз існуючих користувацьких інтерфейсів для систем зберігання даних, досліджено їх недоліки та переваги.

2.1 Поняття інтерфейсу

В інформаційних технологіях інтерфейс — це те, що розроблено як інформаційний блок, з яким можна взаємодіяти. Він може включати комп’ютерну мишу, екран, клавіатуру та зовнішній вигляд робочого столу. Це також те, за допомогою чого як користувач спілкується з додатком або веб-сайтом. Велика залежність багатьох компаній від веб-додатків і мобільних додатків призвела до зростання пріоритетності інтерфейсу, з метою поліпшення комфорту при використанні певного додатку.

Одним з найпопулярніших варіантів інтерфейсу на сьогодні є веб-інтерфейс.

Веб-інтерфейс це різновидність графічного інтерфейсу користувача для зв'язку між користувачем комп'ютера і веб-службою або веб-додатком. Веб-додаток або веб-сервіс можуть перебувати на сервері, підключеному до мережі Інтернет або локальної мережі. У багатьох випадках до веб-служби можна отримати доступ через веб-браузер або через плагін на веб-сторінці наприклад — через Flash Player або аплет Java. Веб-сервіс також може бути використаний користувачем програми на планшетному комп'ютері, смартфоні або смарт-телевізорі.

Користувач веб-служби не обов'язково повинен бути живою людиною, але може також бути ботом або вбудованою системою. Передача даних через веб-інтерфейс зазвичай відбувається через протокол передачі гіпертексту (HTTP), хоча наразі все

частіше від протоколу HTTP відмовляються но корить більш безпечного протоколу HTTPS.

Основною перевагою веб-інтерфейсу є те, що для його використання необхідно встановлювати певні додатки, достатньо мати лише браузер.

2.2 Існуючі рішення

Насьогодні одним з найпопулярніших сервісів зберігання даних є Гугл диск (Google Drive) від компанії Google, що було створено 24 червня 2012 року і підтримується дотепер.

Веб-інтерфейс для Google Drive було написано з використанням фреймворку Angular, що є розробкою компанії Google.

Розглянемо його переваги та недоліки.

Серед переваг можно відмітити наступне:

— сервіс є частково безкоштовним, користувач має в своєму розпорядженні 15 Гігабайт пам’яті безкоштовно, інше — за окрему доплату;

— є прив’язка до аккаунту Google, тобто немає потреби окремо реєструватися для того щоб скористатися цим сервісом.

Але не дивлячись на всі переваги Google Drive має наступні недоліки:

— немає зручного інструменту для інструменту для керування групами користувачів;

— якщо до певного файлу чи папки було відкрито доступ группі, то відсутня можливость понизити рівень доступу до певного ресурсу окремим членам цієї групи;

— кожен користувач, що має доступ до файлу чи папки може надати доступ іншим користувачам системи, що не завжди є прийнятним з причин безпеки;

— відсутні обмеження на вхід в систему в цілому, що не зовсім підходить, для конкретної організації, такої як кафедра.

Висновки до розділу

В даному розділі було розглянуто поняття інтерфейсу, та його підвиду веб-інтерфейсу, проаналізовано існуючі програмні рішення, виявлено їх недоліки та

переваги.

3. ЗАСОБИ РЕАЛІЗАЦІЇ ПРОГРАМНОЇ СИСТЕМИ

Проаналізувавши данне завдання в якості інтерфейсу для системи було вирішено вибрати веб-інтерфейс.

В якості середовища розробки було обрано IntelliJ IDEA від компанії JetBrains.

Для розробки користувацького інтерфейсу було обрано фреймворк Angular від компанії Google.

Для верстки було застосовано бібліотеку Angular Material.

Серверна частина була розроблена на мові програмування Java, з використанням фреймворку Spring, В якості СУБД було застосовано PostgreSQL.

В якості хмарного середовища для розгортання системи було обрано платформу Heroku

3.1 Середовище розробки Intellij IDEA

Середовище розробки Intellij IDEA спочатку було розроблено для мови програмування Java, але потім було додано підтримку таких популярних мов програмування як JavaScript, TypeScript і т.д. Наразі Intellij IDEA містить всі необхідні інструменти розробки як для клієнтської так і для серверної частини. Також вона включає в себе засоби для перегляду більшості відомих СУБД, містить інтеграцію з системами контролю версій, зокрема з такою як Git, також містить засоби для зручного використання таких інструментів як Maven та npm. Окрім того є можливість додавати різні плагіни, що значно полегшують та пришвидшують розробку програмного коду. Також Intellij IDEA має можливість автодоповнення, що значно пришвидшує роботу, а також підвищує точність і допомагає уникати помилок при написанні коду.

Ще однією вагомою перевагою Intellij IDEA є те, що при допущенні помилки, або написання коду, що може потенційно містити помилку, середа розробки не тільки вкаже на помилку, та її місце, а ще й запропонує варіанти її вирішення. Наприклад у випадку якщо програміст намагається використати якийсь метод з певної бібліотеки, але ця бібліотека не підключена до проекту, або ж підключена, але не імпортована в

конкретний файл, в якому програміст намагається її використати, Intellij IDEA запропонує цю бібліотеку підключити, або ж імпортувати.

При розробці даного програмного продукту застосовувалась найновіша на той момент версія 2019.1.1

3.1.1 Якість коду

При написанні коду важливо не лише розробити основний функціонал, але й зробити це якісно, відповідно сучасних стандартів, дотримавшись при цьому загальноприйнятих практик. Вбудований в Intellij IDEA засіб для роботи з TypeScript допоможе відслідкувати порушення норм та загальноприйнятих практик, та автоматично виправити код, або ж якщо є декілька варіантів виправлення, то програмісту буде запропоновано вибір, який з варіантів обрати.

Те саме стосується випадків коли існує більш короткий та простий спосіб, написати те, що програміст написав більш довгим, чи менш зручним способом. В такому випадку цей фрагмент коду буде підсвічено сірим кольором, та при натисканні комбінації клавіш Alt + Enter буде запропоновано замінити цей фрагмент на більш простий.

3.1.2 Усунення помилок

Піктограми на передній панелі курсору вказують, де ви зробили помилку і допомагаєте виправити її. У стандартному інтерфейсі Intellij IDEA, ліворуч від смуги прокручування, розробник має вертикальну лінію, що вказує на помилку, і, звичайно, ви маєте можливість швидко переміститися на це місце.

3.1.3 Автодоповнення та навігація

Середовище розробки Intellij IDEA надає зручні інструменти для автодоповнення та навігації по коду програми, та по підключеним стороннім бібліотекам. Наприклад натиснення на строку коду де викликається певний метод, чи клас з натиснутою при цьому клавішею Ctrl, перенесе нас до початкового коду цього методу, або, якщо такої можливості немає, то до його декомпільованої версії. також Intellij IDEA надає нам три

інструменти для пошуку інформації в контексті:

— пошук введеної фрази в середині поточного файлу;

— пошук введеної фрази по всій програмі;

— пошук за назвою файлу.

3.2 Загальна архітектура системи

В якості загальної архітектури системи було обрано стандартну трирівневу архітектуру.

Трирівнева архітектура — це тип архітектури програмного забезпечення, що складається з трьох "рівнів" логічних обчислень. Зазвичай такий підхід використовуються в тих програмах, що являють собою клієнт-серверну систему. Трирівневі архітектури мають багато переваг для виробничих та розробницьких середовищ, дозволяючи зручно модифікувати інтерфейси користувача, бізнес-логіку та шари зберігання. Це дає командам розробників більшу гнучкість, оскільки вони можуть оновлювати певну частину програми незалежно від інших частин. Така підвищена гнучкість може покращити загальний ринковий час і скоротити час циклу розробки, що дозволяє командам розробників замінювати або модернізувати незалежні шари, не впливаючи на інші частини системи.

Наприклад, користувацький інтерфейс веб-додатку може бути перероблений або оновлений без впливу на функціональну бізнес-логіку і доступ до даних. Ця архітектурна система часто ідеально підходить для вбудовування та інтеграції стороннього програмного забезпечення в існуючу програму. Ця гнучкість інтеграції робить його ідеальним для вбудовування аналітичного програмного забезпечення в існуючі програми, тому його часто використовують постачальники програмного забезпечення для вбудованої аналітики. Схему такої архітектури наведено на рисунку 3.2

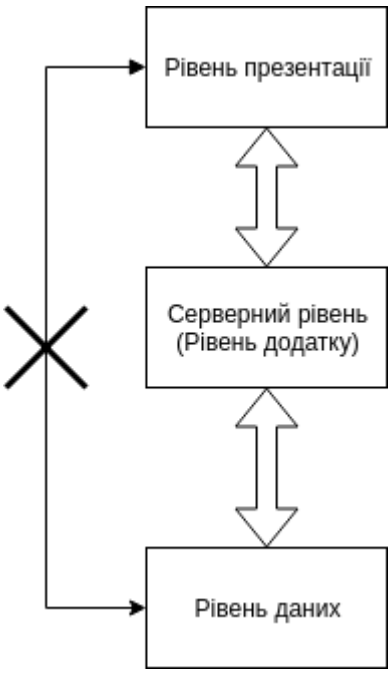

Рисунок 3.2 — трирівнева архітектура програми.

Розглянемо детальніше усі рівні трирівневої архітектури:

Рівень презентації — це перший рівень в трирівневій системі. Він являє собою користувацький інтерфейс. В нашому випадку цим рівнем є веб-інтерфейс, доступний через веб-браузер. Відповідно в нашому випадку цей рівень базується на HTML, CSS, та JavaScript. Але, теоретично він може базуватися і на інших технологіях, якщо використовується інший варіант інтерфейсу (не веб). Також цей рівень зазвичай спілкується з серверним рівнем, або ж з рівнем додатків за допомогою API.

Серверний рівень (рівень додатку) — прикладний рівень містить функціональну бізнес-логіку, яка керує основними функціями програми. В нашому випадку написаний на мові програмування Java. Надає API для рівня презентації, також комунікує з рівнем даних.

Рівень даних — на цьому рівні зберігаються дані. Складається з бази даних / системи зберігання даних і рівня доступу до даних. В даній системі застосовано PostgreSQL.

Використання трирівневої архітектури надає багато переваг, включаючи швидкість, продуктивність і доступність. Як згадувалося раніше, шляхом зміни різних рівнів додатків, команди розробників можуть розробляти та вдосконалювати продукт швидше, ніж якби вони створювали все в єдиному прошарку даних, оскільки один

рівень може бути оновлений чи змінений не впливаючи на інші рівні (або ж впливаючи мінімально). Це також сприяє підвищенню ефективності розвитку продукту, надаючи командам можливість зосередитися на ключових компетенціях. Тобто програмісти будуть мінімально залежати один від одного, і можуть принести найбільшу користь згідно до своєї компетенції.

Масштабованість - ще одна велика перевага трирівневої архітектури. Маючи три різні рівні, можна змінювати розмір цих рівнів, незалежно один від одного, згідно до ваших вимог. Наприклад, якщо ви отримуєте велику кількість запитів на сервер, але більшість з них не потребують запитів в базу даних, ви можете збільшити потужність, або кількість серверів додатку, не чіпаючи при цьому сервери баз данних. Таким чином можна самостійно збалансувати навантаження кожного рівня і поліпшити необхідні ресурси. Крім того, незалежність, утворена різними рівнями модуляції, надає багато можливостей. Наприклад, ви можете розміщувати веб-сервери в публічних, чи приватних хмарних середовищах, а рівень даних можна розміщувати локально.

Використовуючи різні шари, ви також можете підвищити надійність і доступність, надаючи різні частини програми на різних серверах і використовуючи кешування. У випадку якщо один з серверів перестане працювати використовуючи кешування можна дати можливість системі хоча б частково продовжити роботу, а також зменшити час простою системи, а разом із цим і втрати, від цього простою, поки проблему не буде вирішено.

3.3 Архітектура клієнтської частини

В якості архітектури для клієнтської частини було обрано концепцію односторінкового додатку.

У минулому, коли браузери були набагато менш ефективними, ніж сьогодні, а JavaScript вважався досить слабким інструментом, кожна сторінка надходила з сервера. Щоразу, коли користувач щось натискав, на сервер надсилався новий запит, і браузер завантажував нову сторінку.

Лише деякі продукти працювали інакше і експерементували з іншими інноваційними підходами.

Сьогодні все частіше використовується поляризована сучасними фреймворками, такими як Angular, концепція односторінкового додатку.

Ідея полягає в наступному: ви завантажуєте код програми (HTML, CSS, JavaScript) лише один раз, при першому запиті. Надалі вся взаємодія відбувається на рівні браузера, і замість того, щоб відправляти запит на сервер, аби отримати нову сторінку, браузер запитує у сервера лише JSON, а сторінка повністю ніколи не видаляється. Ця поведінка більше нагадує класичний комп'ютерний додаток.

Односторінкові програми створюються за допомогою JavaScript (або TypeScript, що є агрегованою формою JavaScript) і запускаються в браузері.

Як приклад відомих програм, що використовують концепцію односторінкового додатку можна навести наступні:

— Gmail

— Google Maps

— Facebook

— Twitter

— Google Drive

З точки зору користувача односторінковий додаток є набагато швидшим за класичний варіант, адже тепер ви можете отримати швидку відповідь, а не чекати поки сервер сформує сторінку а браузер її обновить.

З точки зору сервера цей підхід не тільки більш швидкий, але й потребує менших ресурсів, адже при такому підході розробники серверної частини можуть повнісю зосередитися на наданні ефективного API, замість створення шаблонів, на стороні сервера.

Як результат, він ідеально підходить для мобільних додатків без зміни API, оскільки ви можете повністю скористатися існуючим кодом на стороні сервера.

Програми з однією сторінкою легко конвертувати в розширені веб-програми, що дозволяють використовувати локальний кеш та працювати навіть якщо користувач знаходиться в режимі офлайн (щонайменше надати коректне повідомлення про помилку,).

Односторінковий додаток є найкращим рішенням, якщо вам не потрібно

оптимізувати пошукову систему, тобто для тих додатків, робота з якими відбувається вже після авторизації.

Односторінковий додаток — ідеальне рішення для роботи в команді. Програмісти, серверної частини, можуть зосередитися лише на створенні API, а розробники клієнтсякої частини можуть повністю зосередитися на створенні користувацького інтерфейсу, лише інтегруючись з наданим API.

3.3.1 Фреймворк Angular

Наразі одним з найвідоміших фреймворків є Angular. Він спеціально був розроблений для створення односторінкових додатків.

Фреймворк Angular було побудовано за ідеєю MVC - Model View Controller (модель, представлення, контролер), хоча автори структури часто називали її Model-View-* або Model-View-Whatever (модель, представлення, що-небудь).

Ідея фреймворку полягала у тому, щоб розділити логіку від маніпуляцій з DOM, також він був націлений на динамічне оновлення сторінки. Однак, це не було надто нав’язливо, тобто є можливість керувати лише частиною сторінки за допомогою Angular. Цей фреймворк має багато зручних функцій, що значно спрощують роботу з користувацьким інтерфейсом, а також дозволяють швидко та легко створювати односторінкові додатки [1].

Зокрема було впроваджено цікаву концепцію зв’язування даних, коли дані на екрані автоматично оновлюються, коли змінюються дані моделі і навпаки. Також було впроваджено ідею директив, що дозволяє створювати власні HTML теги які запускаються за допомогою JavaScript. Тобто задачею програміста є за допомогою JavaScript описати логіку, яка буде підв’язана під цей тег.

Іншим важливим елементом була поява ін’єкції залежностей, що дозволяє об'єднувати компоненти додатків таким чином, щоб код міг бути повторно використаний.

Починаючи з другої версії Angular отримав свою теперішню назву (до цього він називався AngularJS).

Однією з важливих особливостей другої версії (як і всіх наступних) стала

можливість розробки одразу на декількох платформах (веб, мобільна версія, та звичайні додатки).

Розглянемо основні переваги Angular:

Фтпгдфк не тільки надає широкий інструментарій, але й задає шаблони, які дозволяють притримуватись стійкого архітектурного дизайну. Правильно розроблений додаток на Angular не створює класів і методів, які важко модифікувати чи тестувати. Код добре структурований і вам не потрібно багато часу, щоб зрозуміти, що відбувається.

Для розробки використовується TypeScript, який засновано на JavaScript стандарту ES6, а отже він дозволяє робити все те саме що і JavaScript, але крім цього надає ще додаткову функціональність таку як статичні елементи, класи, інтерфейси, простори імен, декоратори та багато іншого.

Фреймворк Angular надає багато стандартних елементів, тож позбавляє вас потреби винаходити велосипед. Завдяки Angular у вас є багато функцій для негайного початку створення програм. Ви можете зручно користуватись формами за допомогою FormControl, а також вводити різні правила валідації для них. Angular з коробки надає вам засоби для динамічної обробки HTML-елементів. Також можна легко надсилати різні типи асинхронних HTTP запитів. Окрім того Angular надає засоби для зручної маршрутизації додатку [2].

Компоненти мають низьке зв’язування один з одним. За допомогою ін’єкції залежностей усувається тісний зв’язок між різними компонентами програми, це також дозволяє легко змінювати різні частини коду незалежно від інших.

Всі маніпуляції DOM відбуваються там, де вони повинні відбуватися. Angular допомагає розділити логіку програми від презентації, що значно спрощує розуміння, розробку та подальшу модифікацію програми.

Тестування є важливим елементом. Написаний код слід ретельно перевіряти, для цього Angular підтримує такі інструменти тестування як Jasmine та Protractor.

Angular підтримує багато платформних систем, а це означає що їм можно однаково зручно користуватися як для мобільної, так і для звичайної версії.

Цей фреймворк активно підтримується та контролюється, окрім того має велику

кількість користувачів, а також власну екосистему. Це означає, що можна легко знайти багато матеріалів, а також стандартних рішень більшості проблем які можуть виникнути при розробці. Окрім того Angular має багато інструментів від сторонніх компаній [3].

Тобто Angular є не тільки системою, а й платформою, що дозволяє розробникам створювати програми для вебу, мобільних пристроїв і настільних комп'ютерів.

3.3.2 Дизайн клієнтської частини додатку

Для створення та проектування дизайну користувацького інтерфейсу було застосовано бібліотеку Angular Material.

Angular Material є бібліотекою компонентів інтерфейсу для програмістів, що використовують Angular. Компоненти, що надаються Angular Material допомагають легко створювати послідовні, привабливі та функціональні веб-додатки, задовольняючи при цьому сучасні принципи веб-дизайну, такі як портативність, незалежність. До того ж, вони відповідають стандартам Material Design, сформованих та описаних компанією Google.

3.4 Архітектура серверної частини

В якості архітектури серверної частини було застосовано концепцію Model-View-Controller, або скорочено MVC. Схему цього шаблону проектування можна побачити на рисунку 3.4

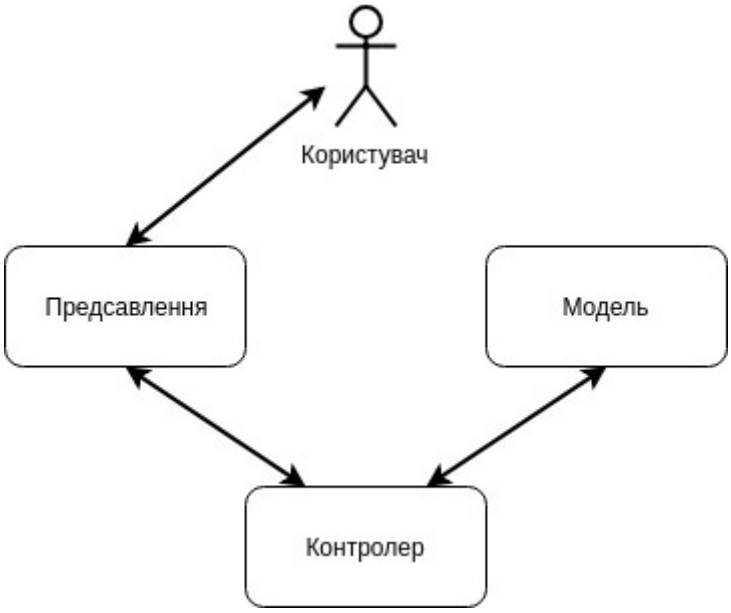

Рисунок 3.4 - шаблон проектування MVC

MVC - це відомий шаблон проектування, а також досить популярний спосіб організації програмного коду. Ідея MVC полягає в розділені коду на секції і розподілення відповідальності за окремі частини програми між цими секціями. Це робиться для того, аби відділити бізнес логіку, від представлення, та зменшити зв’язаність системи. Тобто одна частина відповідає за бізнес логіку та зберігання даних, друга за представлення і третя забезпечує функціонування системи як такої а також робить забезпечує комунікацію перших двох частин програми.

Такий підхід значно спрощує розуміння програми, робить код більш простим, чистим та зрозумілим, до того ж такий код значно легше підтримувати.

Розглянемо тепер всі частини цього патерну більш детально:

— Модель (англ. Model) містить в собі основну бізнес-логіку системи, та відповідає за зберігання даних. Часто саме модель містить в собі всі класи сутності та основні компоненти програми, також відповідає за комунікацію з базою даних, або іншим сховищем.

— Представлення (англ. View) відповідає за пряму комунікацію з користувачем, у веб додатках частіше всього представленням виступає HTML сторінка.

— Контролер (англ. Controller) відповідає за комунікацію моделі та представлення. Також може відповідати за валідацію даних, що прийшли з представлення, перед тим як відправити ці дані для обробки в Модель.

3.4.1 Фреймворк Spring

Для написання серверної частини додатку було застосовано фреймворк Spring.

Фреймворк Spring наразі є найпопулярнішим фреймворком для розробки промислових, і не тільки додатків на мові програмування Java. Мільйони продуктів що використовуються в різноманітних областях використовують цю платформу для створення ефективного, легкого для тестування і розробки коду.

Даний фреймворк має відкритий вихідний код, написаний на Java, та розповсюджується під ліцензією Apache 2.0.

Не дивлячись на достатньо велику функціональність цього фреймворку він є досить легким. Основна його частина займає лише 2 мегабайти пам’яті.

Основні можливості фреймворку Spring можна використовувати для створення будь якого додатку на мові програмування Java. Однак цей фреймворк має багато розширень для розробки веб-додатків на платформі Java Enterprise Edition (скорочено Java EE або J2EE). Основною ціллю фреймворку Spring є полегшення використання J2EE, та сприянню використовування загальних хороших практик програмування з використанням POJO моделі.

Розглянемо основні переваги цього фреймворку:

— Використовуючи Spring розробники можуть використовувати POJO, і немає необхідності використовувати EJB (Enterprise Java Bean). Основна перевага такого підходу полягає в тому що немає потреби у використанні контейнеру для EJB, наприклад серверу додатків, але тим не менш залишається можливість використовувати надійні сервлет-контейнери, наприклад Tomcat, або інший комерційний продукт.

— Фреймворк Spring має добре побудовану модульну структуру, тобто не дивлячись на великий та об’ємний функціонал, та достатньо велику кількість класів, розробник може спокійно ігнорувати не потрібні йому в даний момент, і використовувати лише те, що потрібно.

— Замість того, аби створювати нові рішення уже вирішених проблем, Spring інтегрується з вже готовими технологіями, такими як ORM системи (наприклад Hibernate), Quartz, J2EE, деякі бібліотеки для логування а також ряд технологій для

представлення.

— Тестування програм, написаної з використанням Spring є досить простим, за рахунок ін’єкції залежностей. Адже Spring підставляє потрібний код в залежності від середовища, що дозволяє досить легко використовувати тестові дані, і відділяти їх від тих, що будуть використані, коли додатком будуть користуватися кінцеві користувачі.

— Веб-фреймворк Spring MVC є добре продуманим рішенням для розв’язання задач пов’язаних з вебом.

— Spring надає зручний API, для роботи з винятками, що можуть бути викликані при використанні деяких сторонніх технологій, таких як Hibernate, JDBC і т.д. Spring їх обробляє і віддає нам у вигляді послідовних виключень, що не вимагають перевірки.

— Цей фреймворк надає легкі (у порівнянні з EJB) контейнери для IoC (Inversion of Control), що допомагає значно економити ресурси і є досить вагомою перевагою, при створенні додатків на компьютерах з обмеженими ресурсами процесора та памятью [4].

— Фреймворк Spring надає послідовний та простий інтерфейс управління транзакціями, які також можуть бути змінені як до локальної транзакції (наприклад з однією базою даних) так і до глобальних транзакцій (наприклад з використанням JTA).

Найчастіше фреймворк Spring пов’язують з таким поняттям як ін’єкція залежностей (англ. Dependency Injection, скорочено DI) від інверсії контролю (англ. Inversion of Control, скорочено IoC).

Ін’єкція залежностей є лише одним з варіантів реалізації інверсії контролю, адже інверсія контролю є досить загальним поняттям і може бути досягнута багатьма різними способами.

При написанні комплексного великого додатку на Java, класи цього додатку повинні бути якомога більш незалежними один від одного, для того щоб підвищити можливість повторного використання цих класів, а також надати можливість протестувати їх в модульних тестах незалежно один від одного. Ін’єкція залежностей дозволяє зв’язувати і використовувати ці класи разом, одночасно зберігаючи їх незалежними.

Ін’єкція залежностей відбувається за рахунок контектсу Spring. тобто, якщо клас. Ін’єкція може відбуватися як на рівні конструктора, так і після того як конструктор

спрацює, на рівні сетерів.

Для спрощення роботи з фреймворком Spring компанія Pivotal також розробила Spring Boot — проєкт, що є надбудовою над Spring.

Основним превагами якого є те, що він дозволяє значно спростити написання конфігурації для Spring, переводячи весь процес конфігурації з Java і XML на конфігурацію за допомогою анотацій.

Також, Spring Boot має вбудований сервер Tomcat, а отже додаток не потрібно розгортати окремо.

3.4 База даних

В якості сховища даних було обрано реляційну систему управління базами даних PostgreSQL.

PostgreSQL являє собою потужну систему об’єктно-реляційних баз даних, і є проектом з відкритим початковим кодом. Система розробляється та активно використовується вже достатньо довго, і за цей час встигла зарекомендувати себе як досить надійна система керування базами даних, що може гарантувати збереження цілісності і точності даних.

PostgreSQL працює на більшості відомих операційних систем, таких включаючи Windows та більшість UNIX подібних систем (Linux, Mac OS, Mac OS X, BSD, AIX, SGI-IRIX, HP-UX, Solaris, Tru64). PostgreSQL підтримує тексти, звукові записи, відеозаписи. Також ця СУБД включає в себе програмні інтерфейси для таких відомих мов програмування як Java, Python, C/C++, Perl, Ruby а також підтримує Open Database Connectivity.

Окрім того, що PostgreSQL підтримує більшу частину SQL, він також надає деякі розширені функції, включаючи:

— Комплексні SQL запити;

— Вкладені оператори SELECT;

— Зовнішні ключі;

— Тригери;

— Представлення;

— Транзакції;

— Мультиверсійний контроль задач, що виконуються паралельно;

— Потокове копіювання;

— Гарячий режим очікування.

Крім цього PostgreSQL надає можливість для розширення. Тобто користувач має можливість розширювати функціонал за рахунок додавання нових:

— Типів даних;

— Функцій;

— Операторів;

— Агрегуючих функцій;

— Індексних методів.

Ще однією вагомою перевагою PostgreSQL є те, що це єдина Система керування базами даних, яку можна використовувати безкоштовно, у хмарному середовищі Heroku.

3.5 Хмарне середовище

Одним із ключових завдань даної роботи було те, що система має бути розгорнутою в хмарному середовищі. Розглянемо спершу поняття хмарного середовища та хмарних обчислень.

Хмарні обчислення — це загальне поняття, яке включає в себе широкий спектр послуг, щодо розміщення сервісів в мережі Інтернет. Ці послуги можно умовно поділити на 3 основні категорії:

— Програмне забезпечення як послуга;

— Платформа як послуга;

— Інфраструктура як послуга;

Сама назва “Хмарні обчислення” походить від значку хмари, яким часто забражаловь все, що відноситься до інтернету на різних діаграмах на схемах.

Існує три основні характеристики, що відрізняють хмарний сервіс від звичайного веб-хостингу.

По-перше — хмарний сервіс надається за запитом, в режимі реального часу.

Зазвичай це відбувається швидко та займає хвилину, хоча іноді може займати до однієї години.

По-друге — хмарне середовище є більш гнучким, за традиційний хостинг. Якщо користувачу знадобилося розширити спектр послуг, які він використовує, він має можливість зробити це у будь який момент часу.

По-третє — послуга повністю контролюється провайдером, тим часом як користувач не потребує нічого крім персонального комп’ютера та доступу в Інтернет.

Хмарне середовище може бути приватним, або загальнодоступним. Основна відмінність полягає в тому, що послугами публічного (загальнодоступного) хмарного середовища може скористатися будь-хто в Інтернеті (зараз найбільшим постачальником послуг хмарного сервісу вважається компанія Amazon). В той час як приватне хмарне середовище — це власна мережа, або центр обробки даних, що надає послуги лише для обмеженої кількості користувачів [5].

Незалежно від від типу хмарного середовища, його метою є забезпечення легкого, маштабованого доступу до обчислювальних ресурсів.

3.5.1 Моделі розгортання хмарних обчислень

Послуги приватної хмарної системи надходять з центру обробки даних і доставляються внутрішнім користувачам. Модель такого типу надає основні переваги використання хмарного середовища, такі як універсальність та зручність, зберігаючи при цьому можливість контролю та безпеку, що є загальними для місцевих центрів обробки даних. Послуги для внутрішніх користувачів можуть надаватися як за гроші, так і безкоштовно. Найчастіше приватні постачальники послуг у сфері хмарних обчислень використовують такі технології як VMware та OpenStack.

В публічній моделі постачання хмарних послуг постачальник надає послуги через Інтернет. Публічні сервіси хмарних обчислень продаються за запитом, як правило, за хвилину або годину, хоча часто є можливим заключення довгострокового

контракту, на надання багатьох послуг. Клієнти платять лише за ті ресурси, які ними було використано. Тобто за ресурси процесора, що було використано, використане місце на віртуальному жорсткому диску, зберігання даних, та пропускну здатність. Найвідомішими публічними провайдерами хмарних послуг є: Amazon (AWS), Microsoft (Microsoft Azure) та Google (Google Cloud Platform).

Гібридне хмарне середовище поєднує в собі публічні і приватні сервіси. Також включається оркестровка та автоматизація між ними. Деякі компанії можуть використовувати приватне хмарне середовище, для програм, що мають чутливу інформацію, або ж тих програм, що є критично важливими, в той час як публічна хмара може використовуватись в моменти підвищеного навантаження.

Крім того, зараз серед організацій набирає популярність багаторівнева модель, або використання декількох постачальників сервісу “інфраструктури як послуги”. Це дозволяє одночасно працювати з декількома провайдерами хмарного сервісу, або мігрувати між ними. перевагами цього підходу є те, що це забезпечує стабільність роботи системи, і мінімізує ризик виходу хмарного середовища з ладу. Також це дозволяє для кожної поточної задачі, того провайдера, який може запропонувати найкращий сервіс, для вирішення конкретної задачі (або ж найкращу ціну). Недоліками такого підходу є те, що розробка додатків при цьому значно ускладнюється, адже необхідно підтримувати одночасно декількох провайдерів, а вони можуть мати різні програмні інтерфейси. Але сьогодні використання багаторівневої моделі стає простішим, через те, що список послуг, та відповідні API стають більш схожими та однорідними, частково це відбувається через такі ініціативні проекти як Open Cloud Computing Interface [6].

3.5.2 Переваги використання хмарних обчислень

Хмарні обчислення надають багато переваг, як для компаній, так і для простих користувачів. Розглянемо основні з них:

— Самообслуговування при розгортанні системи: прості користувачі можуть

досить легко розгортати обчислювальні ресурси будь-якого типу навантаження, відповідно до своїх потреб. Це усуває ряд проблем пов’язаних з установкою, керуванням та підтримкою комп'ютерними ресурсами системними адміністраторами.

— Гнучкість: компанії можуть регулювати інфраструктуру згідно до своїх потреб. Тобто якщо компанії потрібно більше ресурсів вона може легко збільшити обчислювальні потужності, у випадку, якщо ресурсів та обчислювальних потужностей потрібно менше їх можна легко зменшити. При цьому немає потреби, вкладати досить великі інвестиції в місцеву інфраструктуру, яка може не використовуватися, або використовуватися лише частково.

— Оплата лише за те, що використовується: використані потужності обчислюються досить детально, а отже користувач системи платить лише а ті ресурси, які він використовує.

— Стійкість до високих навантажень: часто постачальники хмарних сервісів залишають надлишкові ресурси, для того, аби в моменти критичного навантаження система могла це витримати, при цьому не втративши дані. Часто така система розкидається на різні регіони світу.

— Гнучкість до міграції: організації можуть розподіляти перні навантаження на хмарні сервіси, коли в цьому є необхідність, тако є можливість розподіляти навантаження між кількома хмарними платформами. такий сценарій зазвичай використовують або для використання кращих сервісів (тобто використовуватися буде провайдер, який надасть найкращі умови в даний момент часу), або з метою економії коштів.

3.5.3 Типи хмарних сервісів

Хмарні сервіси змінювалися протягом тривалого часу і наразі ці сервіси можна поділити на три окремі категорії: інфраструктура як сервіс, платформа як сервіс, та програмне забезпечення як сервіс.

Розглянемо детальніше кожен із них:

— Інфраструктура як сервіс: яскравим прикладом такого сервісу може виступати AWS, користувач цього сервісу отримує сховище, віртуальний сервер, а

також API, Що дозволяє розподіляти робочі навантаження до віртуальної машини. Користувачі мають визначену ємність пам’яті та можуть запускати, зупиняти, підключатися, та налаштовувати віртуальну машину згідно до своїх потреб. Часто провайдери Інфраструктури як сервісу пропонують малі, середні, великі на надзвичайно великі, а також кастомізовані і оптимізовані згідно до використання пам’яті і ресурсів процесора єкземпляри віртуальних машин.

— Платформа як сервіс: в цьому випадку провайдери надають користувачам засоби розробки, чи певні робочі інструменти, які розміщують на своїх інфраструктурах. Для використання цього сервісу користувачам зазвичай надається API, веб-портал, або програмне забезпечення шлюзу. Платформу як сервіс використовують для загальної розробки програмного забезпечення, і багато провайдерів надають можливість зберігання на розгортання програмного забезпечення після його розробки. Серед відомих прикладів можна навести Force.com, AWS Elastic Beanstalk і Google App Engine.

— Програмне забезпечення як сервіс: це модель хмарного сервісу, що надає готове програмне забезпечення через інтернет. Часто такі програми можуть називати веб-службами. Ідея полягає в тому, що користувач може отримати доступ до програмного забезпечення з будь якого місця, якщо він має доступ до Інтернету. Популярним прикладом такого сервісу є Microsoft Office 365.

3.5.4 Безпека хмарних обчислень

Однією з головних вимог для компаній та підприємств, що використовують, чи планують використовувати хмарні обчислення є безпека даних. Провайдери публічних хмарних середовищ поділяють часто поділяють одну апаратну інфраструктуру, між багатьма користувачами. Отже це середовище вимагає надійної ізоляції обчислювальних ресурсів один від одного. У той же час доступ до загальнодоступного сховища та обчислювальних ресурсів охороняється за допомогою облікових даних користувачів, що гарантує те, що доступ до даних та обчислювальних ресурсів не зможе отримати сторонній користувач.

Багато організацій, пов'язаних складними процесами та стандартами управління, все ще не бажають розміщувати дані або робочі навантаження в публічному хмарному середовищі через побоювання відключення, втрати або крадіжки даних. Однак ці побоювання поступово зникають, оскільки логічна ізоляція продемонструвала себе як достатньо надійна, а додавання шифрування даних та різних інструментів управління ідентифікацією та доступом поліпшило безпеку в публічному хмарному просторі.

3.5.5 Хмарний сервіс Heroku

В якості провайдера по постачанню хмарних послуг було обрано Heroku.

Heroku — це постачальник хмарних сервісів, що надає послуги типу “Платформа як сервіс”, Особливу популярність Heroku набула через свою простоту у використанні та досить широкий спектр пропонуємих сервісів.

Особлива увага приділяється підтримці програм орієнтованих на клієнтів, адже Heroku має досить широкий спектр інструментів для розгортання та розробки програм. Ідея полягає в тому, щоб користувач міг зосередитись на розробці і вдосконаленні своїх додатків, тим часом як Heroku потурбується про всю інфраструктуру, управління ресурсами, розгортання та підтримку апаратних засобів.

Основними перевагами Heroku є:

— Простий та швидкий механізм розгортання додатків за допомогою прив’язки до системи контролю версій. Heroku підключається до системи контролю версій і в налаштуваннях вказується певна гілка (найчастіше це master). Як тільки якісь зміни потрапляють в цю гілку система автоматично збирає та розгортає додаток.

— Heroku містить досить багато додаткових інструментів та ресурсів, такі як бази даних, сторонні програми, методи для інтеграції з іншими системами.

— Добре налагоджені процеси масштабування. Кожен компонент програми може масштабуватся окремо, без впливу на функціональність та продуктивність.

— Ізоляція: кожен процес надійно ізольований один від одного.

— Зручне логування та візуалізація процесів, як з командного рядку, так і з веб-інтерфейсу Heroku можно досить легко подивитися логи кожного процесу,

відслідкувати його поточний стан.

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Розроблений продукт складається з двох основних частин: сервеної та користувацької, для збірки всього проекту використовується інструмента для збірки проєктів maven. Структуру проекту можна побачити на рисунку 4.1.

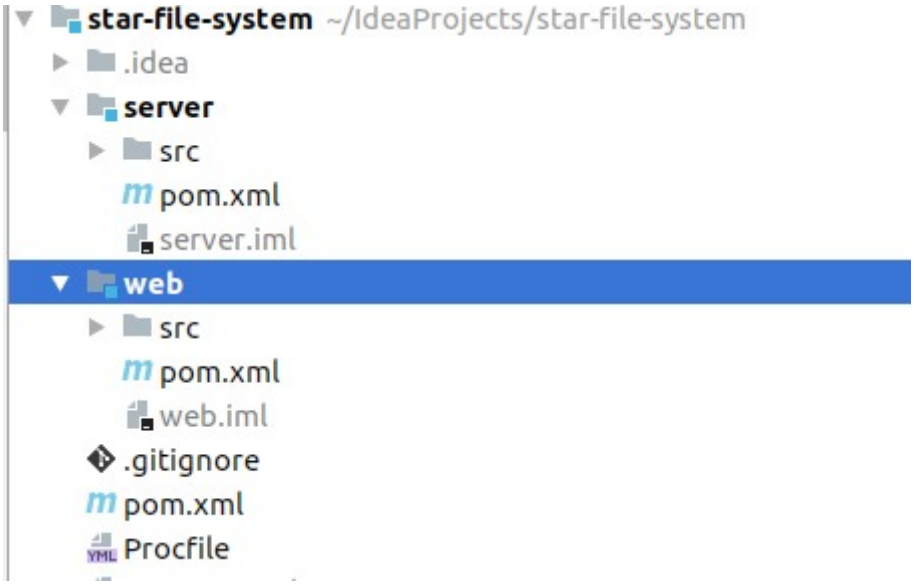

Рисунок 4.1 — Структура проекту.

В головній папці проекту міститься файл pom.xml, що містить інструкції для Maven по збірці проекту. Також є Procfile, що описує порядок дій для розгортання системи на Heroku.

Основним, на що варто звернути увагу є папки “server” та “web”. В папці “server” знаходиться весь програмний код серверної частини, в папці “web” весь код клієнтської частини. Також варто зазначити, що обидві ці папки містять свій власний файл pom.xml, що містить інструкції по тому як збирати кожний окремий модуль.

Як було зазначено вище, збірка всього проекту,відбувається за допомогою Maven, але збірка компонентів Angular, що знаходиться в папці web виконується за допомогою npm. Для того щоб це зв’язати використовується інструмент під назвою “frontend-maven-plugin”, він дозволяє записати у файл pom.xml команди npm, які мають бути виконані, для того аби зібрати компоненти Angular.

Розглянемо детальніше частину коду, що відповідає за користувацький інтерфейс. Більш детальну структуру можна побачити на рисунку 4.2.

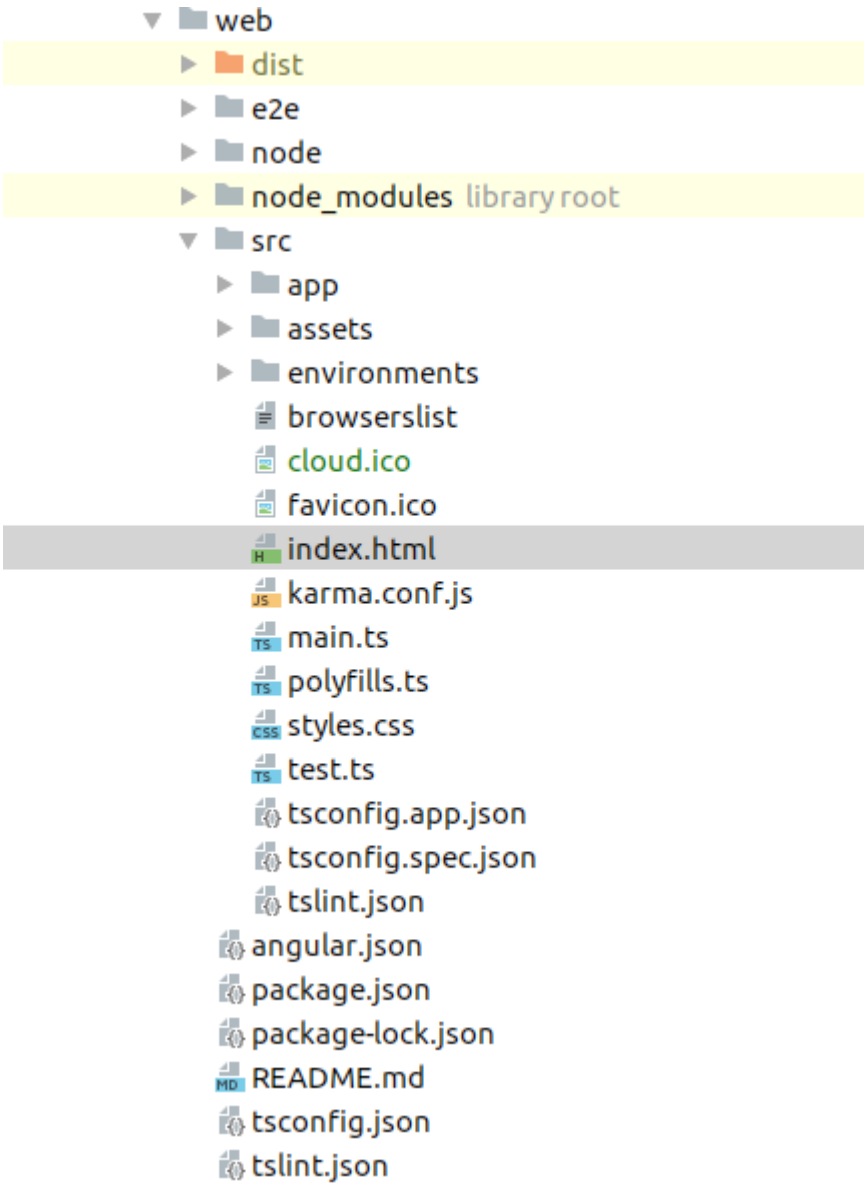

Рисунок 4.2 - вміст папки web

Тут варто звернути увагу на такі основні компоненти:

— папка dist: містить в собі відкомпільований та зібраний в кілька відносно невеликих HTML, CSS та JavaScript файлів. Саме ці файли і будуть віддані в браузер, коли користувач буде працювати з додатком

— папка node_modules: в цю папку складаються всі підключені в додатку бібліотеки.

— папка src: містить вихідний код програми, всі створені компоненти, та кілька згенерованих при початку роботі файлів таких як index.html, main.ts та інші, вони

містять в собі деякі базові інструкції, які генеруються автоматично, і майже не модифікуються програмістом.

— файл package.json: містить в собі інформацію про підключення бібліотек.

— файл angular.json: містить інструкцію про те, як правильно проводити збірку проекту.

Для полегшення процесу розгортання додатку в хмарному середовищі та економії ресурсів було вирішено не розгортати клієнтську та серверну частину окремо, а зібрати все в єдиний war файл. Фактично такий підхід дає нам використовувати всі можливості Angular, але при цьому розгортати все це як єдиний Java додаток.

Це досягається за рахунок копіювання вмісту папки dist, що створюється Angular, в папку resources, що знаходиться в модулі server і використовується фреймворком Spring. Аби не робити це руками, щоразу як нам потрібно перезібрати проект, а також щоб не перемішувати згенеровані файли з початковим кодом до такого процесу ставляться наступні вимоги:

— Процес копіювання має відбуватися автоматично, на етапі збірки проекту.

— Файли мають копіюватися не в вихідний код а в той, що збирається на серверній частині.

Для досягнення цього використовується maven-resources-plugin.

Ідея полягає в наступному: модуль web підключається в модуль server як залежність, після чого за допомогою maven resource плагіну вміст папки dist копіюється в папку resources.

4.1 Частини користувацького інтерфейсу

Розглянемо більш детально окремі частини користувацького інтерфейсу.

В залежності від ролі користувача в системі, сторінка яку побачить користувач буде відрізнятися.

Якщо користувач не авторизований то він має побачити сторінку з інформацією

про те, що для подальшого користування системою він має пройти процедуру авторизації за допомогою Google.

Після проходження авторизації користувач в шапці сайту має побачити ім’я акаунту, під яким він зараз знаходиться в системі, а також кнопку виходу.

Також зліва він має побачити меню, в якому будуть знаходитися наступні пункти: головна, групи, і якщо даний користувач має права адміністратора, то ще розділ управління користувачами.

Реалізація всіх цих частин була розподілена по компонентам Angular наступним чином.

— Шапка сайту (ім’я користувача, кнопка виходу): AppComponent, тобто головний компонент.

— Сторінка з інформацією про те, що для продовження роботи необхідно авторизуватися (для неавторизованих користувачів): Welcome Component.

— Головний розділ (на ній користувач може бачити доступні йому файли та папки): Main Component.

— Розділ управління групами: Group Component.

— Розділ управління користувачами: Admin Component.

— Меню: Menu Component.

Взаємодія з серверною частиною.

При взаємодії з сервером було використано наступні API:

— Отримання інформації про поточного користувача: GET /user_api/me.

— Отримання поточної ролі користувача в системі: GET /get_roles.

— Отримання інформації щодо всіх файлів та папок що доступні користувачу: GET /api/getAllItems.

— Скачування файлу: GET /api/download.

— Надати доступ до файлу чи папки окремому користувачу: PUT /api/share/{id}.

— Надати доступ до файлу чи папки групі користувачів: PUT /api/share_group/{id}.

— Видалити файл або папку: DELETE /api/deleteItem.

— Створити папку: POST /api/createItem.

— Завантажити файл на сервер: POST /api/upload.

— Отримати список користувачів, що мають доступ до даного файлу чи папки: GET /api/users

— Забрати у користувача доступ до файлу чи папки: DELETE /api/removeAccess/{id}.

— Отримати список груп, до яких користувач належить: GET /groups_api/get.

— Створити групу /groups_api/create.

— Додати користувача в групу: POST /groups_api/add_to.

— Видалити групу DELETE /groups_api/delete.

— Змінити права доступу користувача всередині групи: POST /groups_api/set_permission.

— Видалити користувача з групи /groups_api/remove_from.

— Вийти з групи /groups_api/leave.

5. РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Розроблений програмний комплекс було створено як веб-додаток, а отже система працює в браузерах, що підтримують сучасні веб-стандарти.

5.1 Початок роботи з системою

Так як система вже розгорнута хмарній платформі Heroku, вона не потребує інсталяції, локального розгортання чи інших дій по по запуску чи конфігурації системи.

Користувач може почати роботу після того як він отримав запрошення на електронну пошту. Це означає що його було додано до списку користувачів і тепер він може авторизуватися в системі і почати з нею роботу.

Перейшовши за посиланням користувач опиниться на початковій сторінці, де йому буде запропоновано авторизуватися. Вигляд початкової сторінки можна побачити на рисунку 5.1

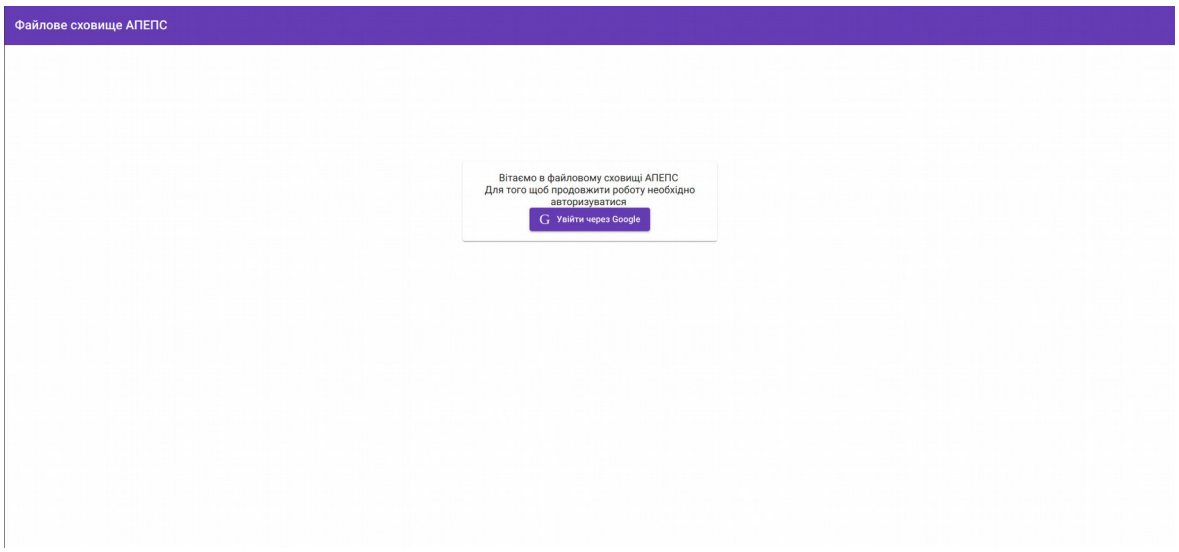

Рисунок 5.1 — початкова сторінка

При натисканні на кнопку авторизації користувачеві відкриється вікно авторизації через Google приклад якого можна побачити на рисунку 5.2

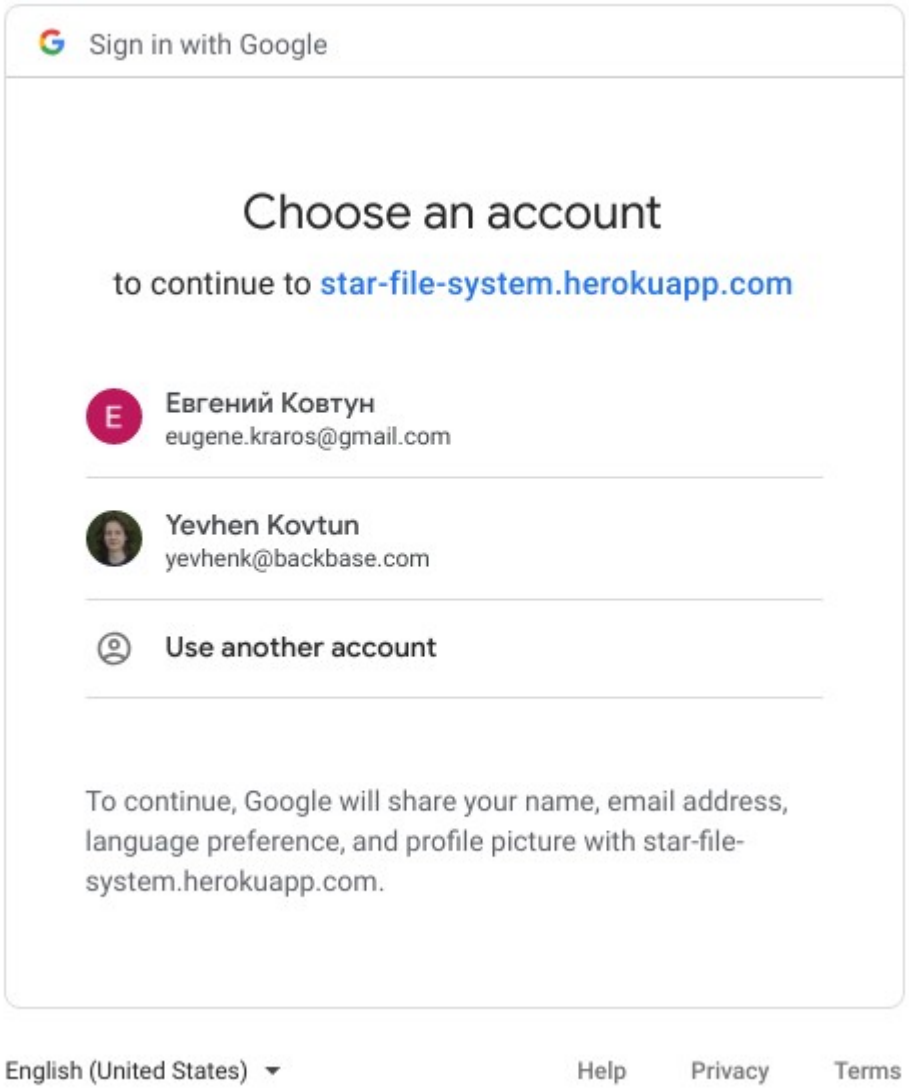

Рисунок 5.2 — вікно авторизації через Google

Після успішного проходження авторизації зліва користувач зможе побачити меню, що складається залежно від ролі користувача з двох, або трьох пунктів: “Головна”, “Групи”, та залежно від того чи є користувач адміністратором, чи ні користувач побачить, чи не побачить розділ “Управління користувачами”. Вигляд меню можна побачити на рисунку 5.3.

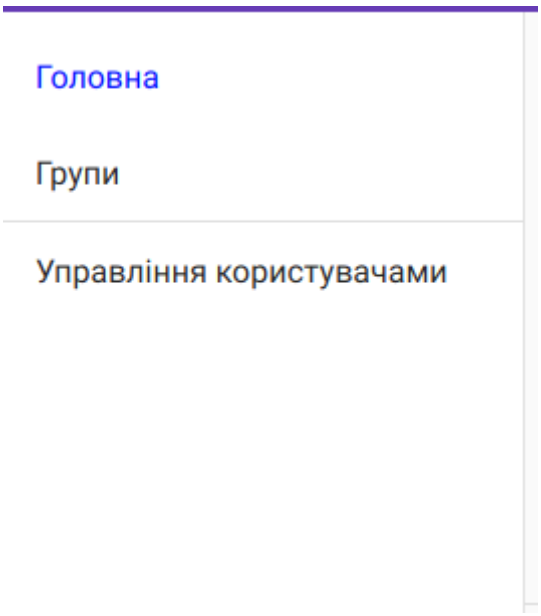

Рисунок 5.3 — меню додатку

5.2 Головна

При переході в розділ “Головна” користувачу відкриється головний розділ додатку, в якому користувач може бачити доступній йому файли та папки. Приклад того, як цей розділ може виглядати можна побачити на рисунку 5.4

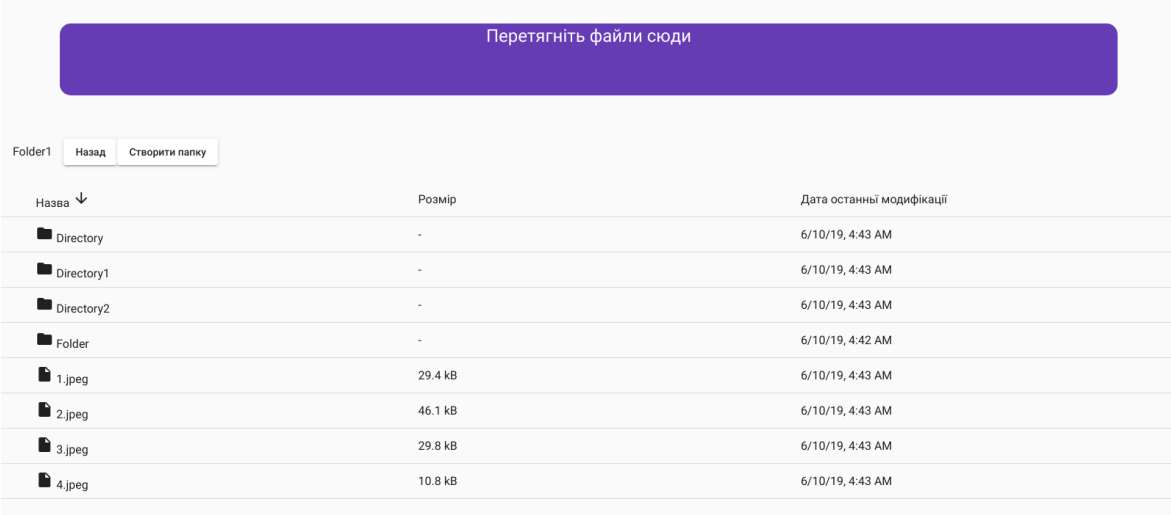

Рисунок 5.4 — головний розділ додатку.

Основною частиною цього розділу є список файлів, що доступний даному

користувачу.

При Подвійному натисканні на папку відбудеться перехід в цю папку, при подвійному натисканні на файл почнеться скачування цього файлу. Зверху над списком вказано назви полів, цього списку. Тут присутні назва файлу чи папки, його розмір, та дата останньої модифікації.

За замовчуванням всі елементи відсортовано в прямому порядку за назвою, але так, що паки завжди йдуть вище за файли. При натисненні на елементи шапки списку зміниться критерій сортування, а якщо повторно натиснути на вже обраний критерій, то зміниться порядок сортування (з прямого на зворотній та навпаки).

Також зверху над таблицею можна побачити назву поточної папки та кнопку повернення назад, ці два елементи відображаються у тому випадку, якщо користувач знаходиться не в кореневій папці проекту. Також поруч із цими елементами можна побачити кнопку створити папку, цей елемент також присутній не завжди. Якщо ми знаходимось у папці, в який ми маємо доступ лише для читання, то ця кнопка буде відсутня.

При натисканні на цю кнопку відкриється діалогове вікно, в якому користувачу буде запропоновано ввести назву папки яку він хоче створити. Приклад такого вікна можна побачити на рисунку 5.5

Рисунок 5.5 — діалогове вікно для створення папки

Зверху вгорі можна побачити зону для перетягування файлів. Якщо перетягнути файли туди, то користувачу спершу відкриється список файлів, які він перетягнув, і

буде надана можливість завантажити їх в поточну директорію, або видалити з цього списку. Приклад завантаження декількох файлів можна побачити на рисунку 5.6

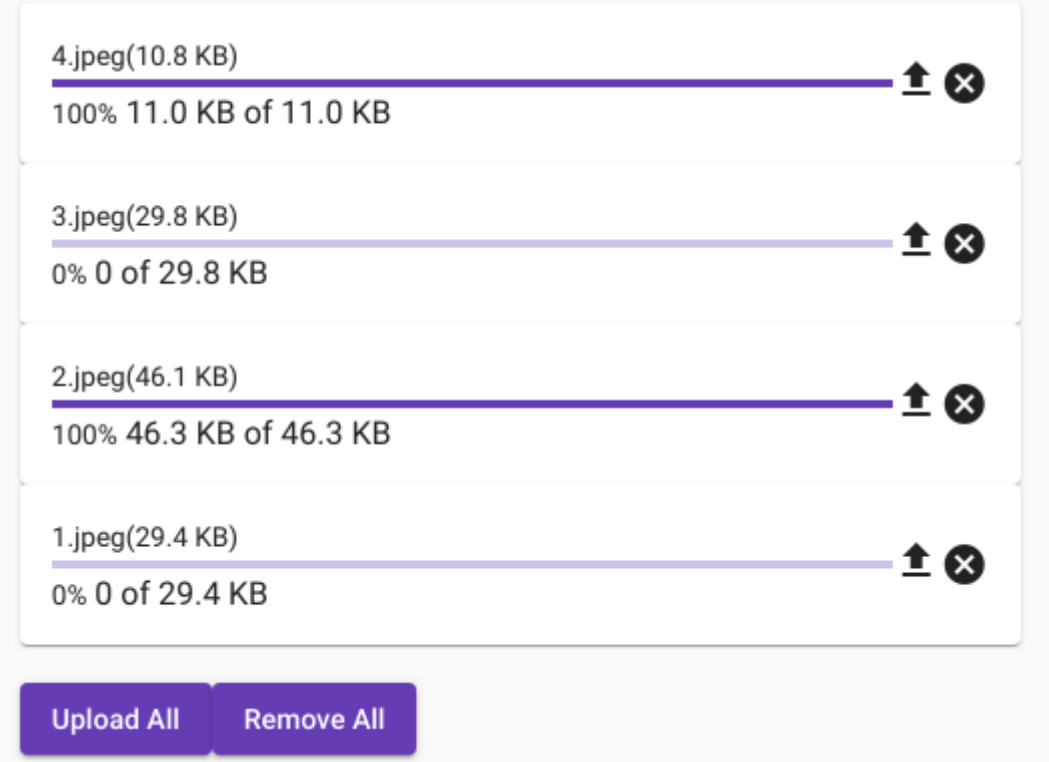

Рисунок 5.6 — завантаження файлів

Також при натисканні правої кнопки миші на чи папці файлі відкриється контекстне меню, приклад якого можна побачити на рисунку 5.7

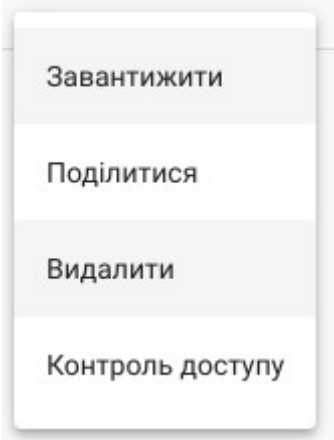

Рисунок 5.7 - контекстне меню єлемента

Список опцій в цьому меню є змінним, адже такі пункти як поділитися, видалити, та контроль доступу доступні лише в тому випадку, якщо користувач має права власника файлу чи папки, а кнопка завантажити присутня лише для файлів.

При обранні пункту меню “Поділитися” відкриється діалогове вікно, приклад якого можна побачити на рисунку 5.8

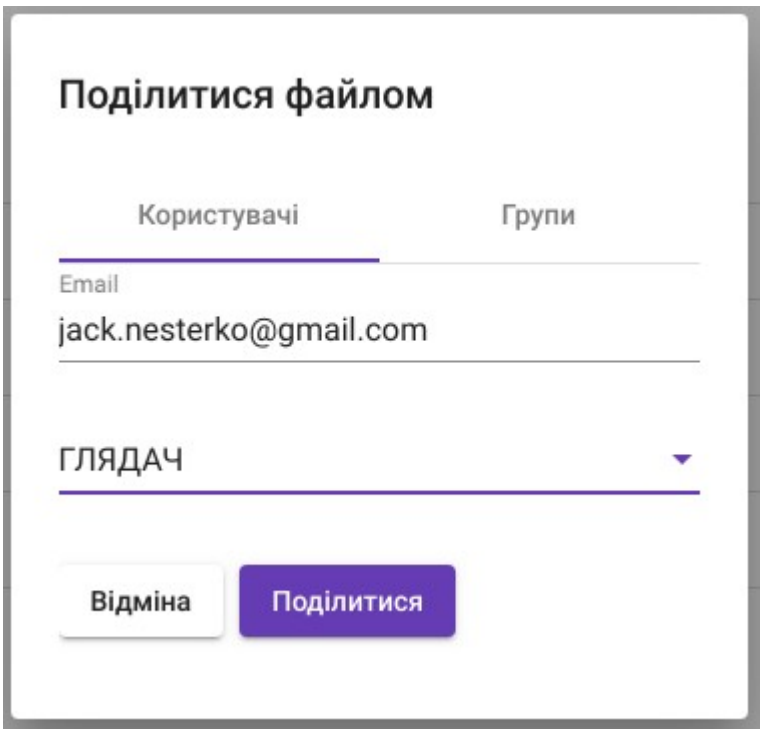

Рисунок 5.8 — діалогове вікно “Поділитися файлом”

В цьому меню є 2 опції: поділитися з користувачем, і поділитися з групою. Якщо ми хочемо поділитися файлом із глядачем, нам необхідно ввести адресу його електронної пошти, та вибрати рівень доступу до даного файлу чи папки. Для більшої зручності інтерфейсу в цьому вікні є функція автодоповнення, що допомагає користувачеві швидко ввести потрібний email і не допустити при цьому помилок. Щодо можливості поділитися з групою, принцип залишається той самий, але замість електронної пошти нам треба ввести назву групи, якій ми хочемо надати доступ.

Після того як ми надали певному користувачеві або групі користувачів доступ до певного файлу чи папки варто розглянути наступний розділ меню — “Контроль доступу”. При виборі цього пункту меню нам відкриється вікно, в якому ми можемо бачити хто має доступ до цього файлу чи папки, а також рівень цього доступу. Приклад цього вікна можна побачити на рисунку 5.9

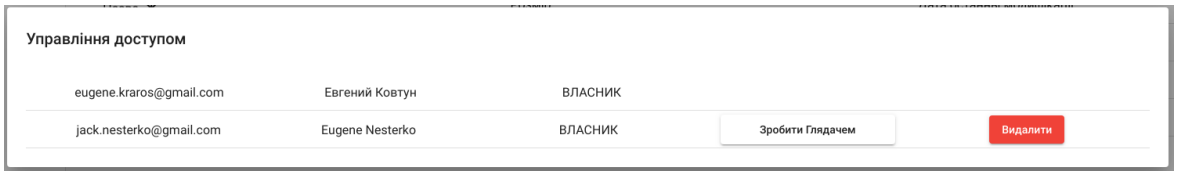

Рисунок 5.9 — Управління доступом до файлу чи папки.

Тут ми можемо бачити список користувачів, що мають доступ до цього файлу чи папки, їх електронну пошту, ім’я та рівень доступу до цього файлу. Крім того є кнопка

для зміни рівня доступу, яка залежить від поточної ролі певного користувача, тобто якщо він є власником файлу, то з’явиться кнопка зробити глядачем, а якщо він є глядачем, то кнопка буде “Зробити власником”. Також є кнопка видалити, яка видалить користувача зі списку тих, хто має доступ до цього файлу чи папки.

При натисканні кнопки видалити відкриється діалогове вікно, в якому користувача запитають, чи він впевнений, що хоче видалити обраний файл чи папку. Приклад цього вікна можна побачити на рисунку 5.10

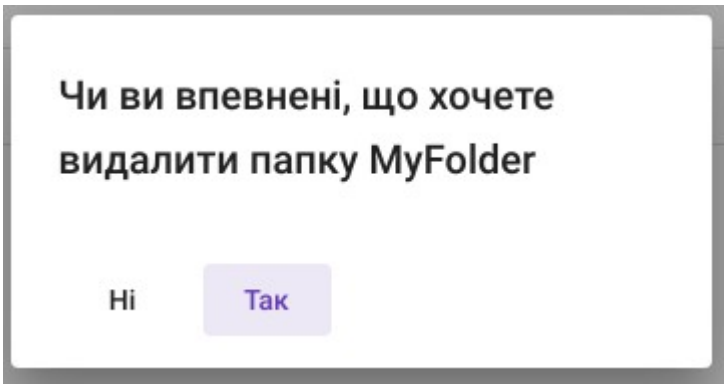

Рисунок 5.10 — діалогове вікно при видаленні папки

Це зроблено для того, аби попередити можливість випадкового видалення папки чи файлу, а отже і можливої втрати цінних даних.

5.3 Групи

Наступним пунктом меню є “Групи”.

При переході туди відкриється меню управління групами де зліва буде розташовано меню з назвами груп, а зправа знаходиться список учасників конкретної групи, який з’являється після того як користувач вибере певну групу клікнувши на неї. Приклад цього меню можна побачити на рисунку 5.11

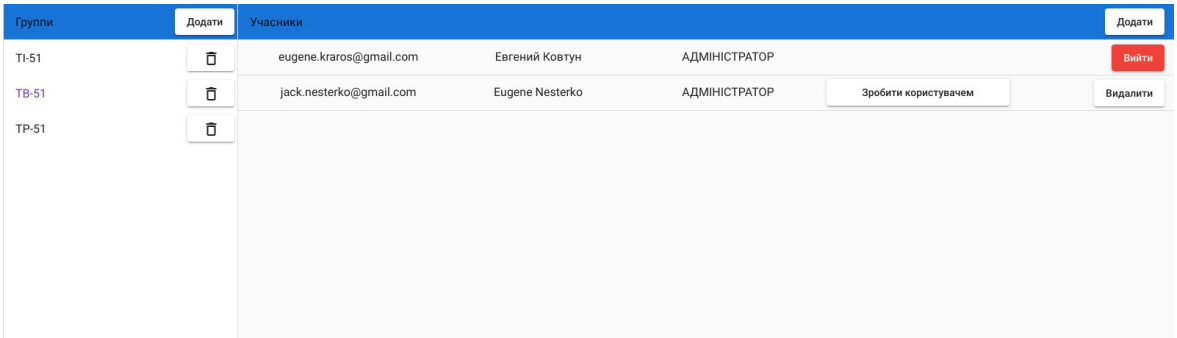

Рисунок 5.11 — розділ “Групи”

В шапці цього розділу біля напису “Групи” є кнопка додати, при натисканні на яку відкриється діалогове вікно, де користувачеві буде запропоновано ввести назву нової групи. Приклад цього вікна можна побачити на рисунку 5.12.

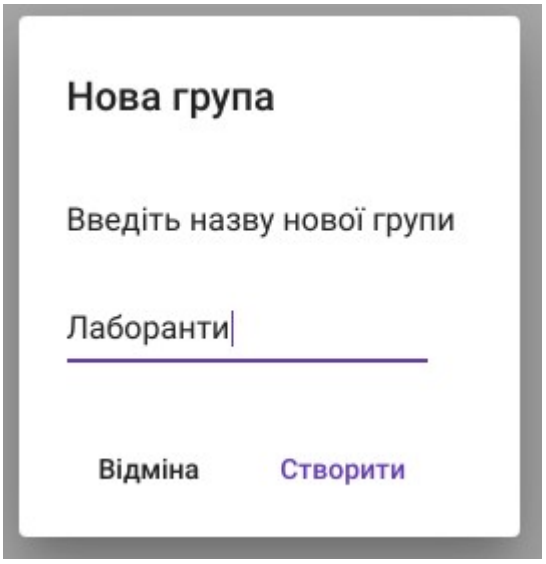

Рисунок 5.12 — діалогове вікно для створення нової групи.

При кліку на группу зліва відображається список учасників цієї групи, при чому поточний користувач завжди буде відображатись першим в списку. Якщо поточний користувач має права Адміністратора групи, то в нього з’явиться можливість додавати учасників в цю групу, видаляти учасників з групи, або ж змінювати їх роль в групі. Важливим моментом є те, що навпроти поточного користувача є кнопка вийти з групи. Якщо користувач є простим учасником групи, ця кнопка відображається завжди, якщо поточний користувач є адміністратором групи, то ця кнопка з’явиться лише в тому випадку, якщо в групі є ще хоча б один адміністратор. При додаванні учасника в групу з’явиться діалогове вікно, яке можна побачити на рисунку 5.13.

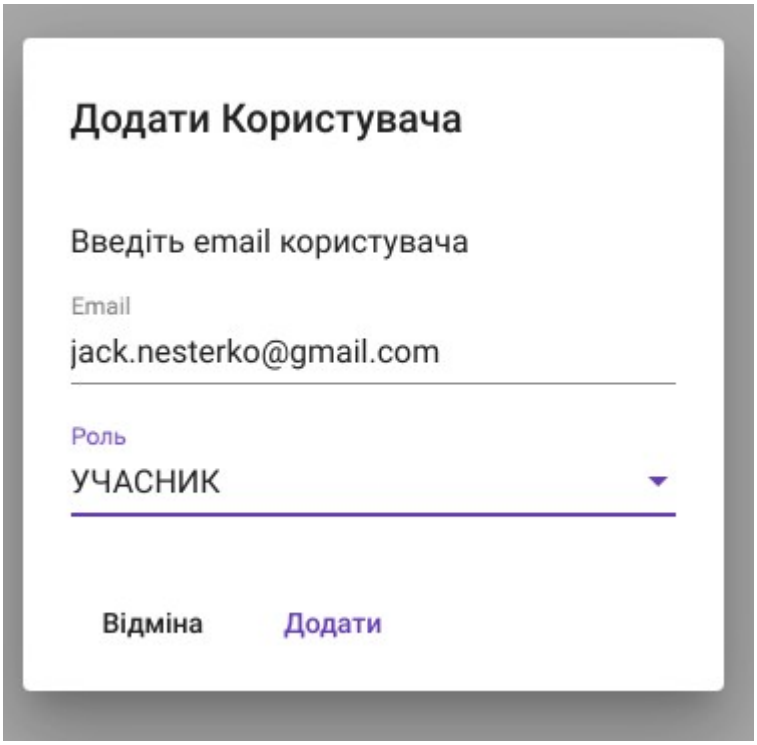

Рисунок 5.13 — діалогове вікно для додавання користувача в группу.

В цьому діалоговому вікні потрібно ввести email користувача, та вибрати його роль в групі. Для спрощення вводу електронної пошти є функція автодоповнення.

5.4 Управління користувачами

Останнім пунктом меню є управління користувачами. До цього пункту меню мають доступ лише адміністратори системи. В цьому розділі міститься функціонал для запрошення, видалення, та модифікації ролі користувача в системі. Приклад цього Розділу можна побачити на рисунку 5.14.

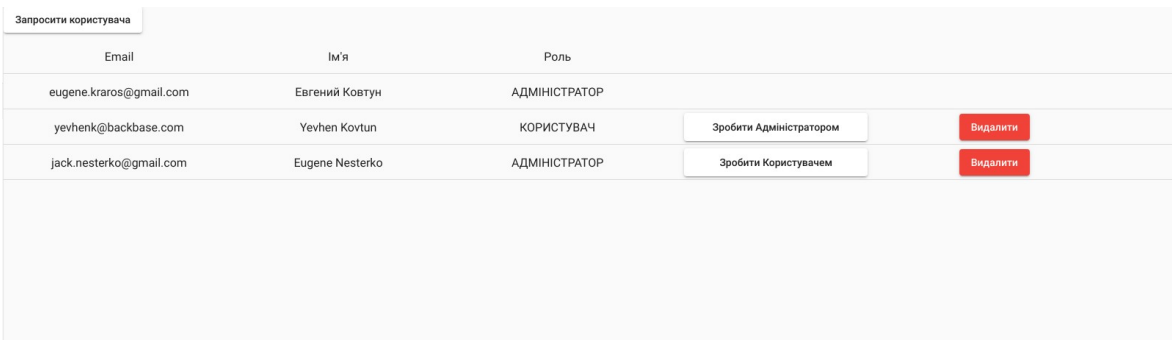

Рисунок 5.14 — розділ управління користувачами.

В цьому розділі представлено список існуючих користувачів системи, їх email, Ім’я та роль в системі, крім того є можливість змінити роль користувача та видалити його з системи. Є також кнопка запросити користувача, при натисканні на яку відкриється діалогове вікно, приклад якого можна побачити на рисунку 5.15

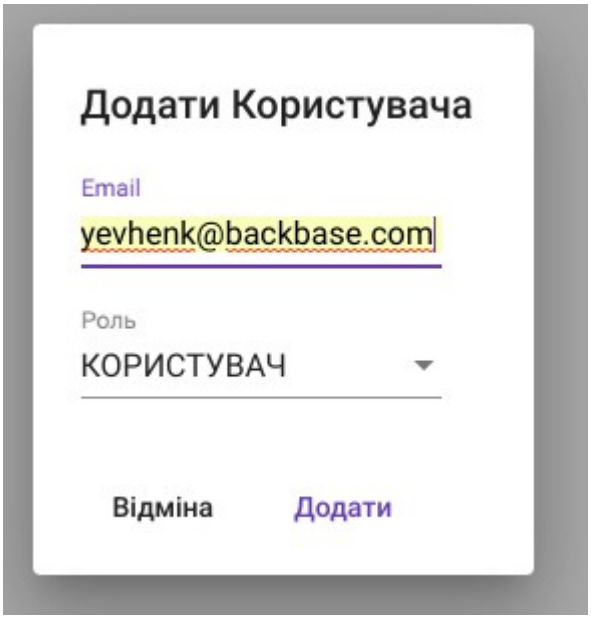

Рисунок 5.15 — діалогове вікно запрошення користувача.

При запрошенні користувача в систему йому на вказану електронну пошту буде надіслано лист, з посиланням на додаток, та інформацією про те, що його було запрошено в систему.

ВИСНОВКИ

В результаті виконання даної роботи було розроблено користувацький інтерфейс, для кафедрального хмарного сховища.

Інтерфейс було написано з використанням таких мов як HTML, CSS, TypeScript, з використанням фреймворку Angular, та бібліотеки Angular Material.

Розроблена система надає можливість зберігати дані у вигляді файлів та папок, надавати доступ до цих файлів чи папок окремим користувачам системи та групам користувачів.

Було реалізовано авторизацію через Google.

Також було створено методи для запрошення користувачів в систему, методи управління користувачами в системі.

Систему було розгорнуто на хмарній платформі Heroku, що надає можливість відразу використовувати систему.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. А. Фримен -Angular для профессионалов, 2018. 256 ст.

2. Флэнаган, Дэвид JavaScript. Карманный справочник / Дэвид Флэнаган. - М.: Вильямс, 2015. - 320 с.

3. Метт Фрісбі. Angular. Сборник рецептов 2016.114ст.

4. Johnson R. Professional Java Development with the Spring Framework / R. Johnson, J. Hoeller, A. Arendsen, et al. – Wrox Press, 2005. – 672 p.

5. Риз, Джордж Облачные вычисления / Джордж Риз. - М.: БХВ-Петербург, 2011. - 288 с.

6. Фингар, Питер Dot.Cloud: облачные вычисления - бизнес-платформа XXI века / Питер Фингар. - М.: Аквамариновая Книга, 2011. - 256 с.

ДОДАТОК А

Створення користувацького інтерфейсу для доступу до кафедрального хмарного сховища.

Специфікація

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ51152_19Б

Аркушів 1

Київ 2019

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ51152_19Б	Записка.doc	Пояснювальна записка
Компоненти		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ51152_19Б 12-1	main.component.ts	Основні компоненти
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ51152_19Б 13-1	Додаток В.doc	Опис програмного модуля

ДОДАТОК Б

Створення користувацького інтерфейсу для доступу до кафедрального хмарного сховища.

Текст програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ51152_19Б 12-1

Аркушів 10

Київ 2019

```
import {Component, Inject, OnDestroy, OnInit, ViewChild} from '@angular/core';
import {ActivatedRoute, Router} from "@angular/router";
import {ItemService} from "../item.service";
import {Observable, Subscription} from "rxjs";
import {ItemWithPermission} from "../item-with-permission";
import {MAT_DIALOG_DATA, MatDialog, MatDialogRef, MatMenuTrigger, Sort} from "@angular/material";
import {UserService} from "../user.service";
import {User} from "../user";
import {FormControl} from "@angular/forms";
import {map, startWith} from "rxjs/operators";
import {GroupService} from "../group.service";
import {GroupWithPermission} from "../group-with-permission";
import {Item} from "../item";
import {UserWithItemPermission} from "../user-with-item-permission";
import {AuthService} from "../auth.service";


@Component({
  selector: 'app-main',
  templateUrl: './main.component.html',
  styleUrls: ['./main.component.css']
})
export class MainComponent implements OnInit, OnDestroy {

  id: number;
  items: ItemWithPermission[] = [];
  itemService: ItemService;
  interval: any;
  sub: Subscription;
  visible = false;
  name: string;
  dataSource: any;
  displayedColumns: string[] = ['item.name', 'item.size', 'item.lastModified'];
  sortedItems: ItemWithPermission[] = [];
  private router: Router;
  isReverseSort = 1;
  sortingParameter = 'name';

  contextMenuPosition = {x: '0px', y: '0px'};
  @ViewChild(MatMenuTrigger)
  contextMenu: MatMenuTrigger;
  sort: Sort;
  current: ItemWithPermission;

  constructor(public dialog: MatDialog, private  activatedRoute: ActivatedRoute, itemService: ItemService, router: Router) {
    this.itemService = itemService;
    this.activatedRoute = activatedRoute;
```

```
    this.router = router;
}

onContextMenu(event: MouseEvent, item: ItemWithPermission) {
    console.log('in context Menu');

    event.preventDefault();
    this.contextMenuPosition.x = event.clientX + 'px';
    this.contextMenuPosition.y = event.clientY + 'px';
    console.log(event);
    console.log(this.contextMenuPosition);
    this.contextMenu.menuData = {'itemWP': item};
    this.contextMenu.openMenu();
}

contextMenuDownload(itemWP: ItemWithPermission) {
    console.log(' IN contextMenuDownload');
    this.itemService.downloadItem(itemWP.item.itemId);

}

contextMenuShareWith(itemWP: ItemWithPermission) {
    const addUserRef = this.dialog.open(ShareWithDialogComponent, {
        data: {id: this.id}
    });
    addUserRef.componentInstance.id = itemWP.item.itemId;
}

contextMenuDelete(itemWP: ItemWithPermission) {
    const deleteRef = this.dialog.open(DeleteItemDialogComponent, {
        width: '350px',
    });
    deleteRef.componentInstance.name = itemWP.item.name;
    deleteRef.componentInstance.id = itemWP.item.itemId;
    deleteRef.componentInstance.directory = itemWP.item.directory;
}

retrieveItems() {
    this.itemService.getItems(this.id).subscribe(
        resp => {
            resp.sort((a, b) => {
                if (a.item.directory && !b.item.directory) {
                    return -1;
                }
                if (!a.item.directory && b.item.directory) {
                    return 1;
                }
```

```
            switch (this.sortingParameter) {
              case 'name':
                return compare(a.item.name, b.item.name) * this.isReverseSort;
              case 'date':
                return compare(a.item.name, b.item.name) * this.isReverseSort;
              case 'size':
                return compare(a.item.size, b.item.size) * this.isReverseSort;
              default:
                return compare(a.item.name, b.item.name) * this.isReverseSort;

            }
          });
          if (JSON.stringify(resp) != JSON.stringify(this.items)) {
            console.log('not eq');
            console.log(JSON.stringify(resp));
            console.log(JSON.stringify(this.items))
            this.items = resp;
          }
        }
            )
}

humanReadableSize(size): string {

  if (Math.abs(size) < 1024) {
    return size + ' b'
  }
  let units = ['kB', 'MB', 'GB', 'TB', 'PB', 'EB', 'ZB', 'YB'];
  let i = -1;
  do {
    size /= 1024;
    i++;
  } while (Math.abs(size) >= 1024 && i < units.length - 1);
  return size.toFixed(1) + ' ' + units[i];
}

ngOnInit() {
  this.id = this.activatedRoute.snapshot.params['id'];
  if (this.id) {
    this.itemService.getCurrentFolder(this.id).subscribe(
      data => this.current = data
    );
  } else {
    this.current = null
  }
  this.sub = this.activatedRoute.params.subscribe(params => {
```

```
      this.id = params['id'];
      if (this.id) {
        this.itemService.getCurrentFolder(this.id).subscribe(
          data => this.current = data
        );
      } else {
        this.current = null
      }
      this.retrieveItems();
    });
    this.retrieveItems();
    this.interval = setInterval(() => {
      this.retrieveItems();
    }, 5000);

    console.log(this.id);
}

ngOnDestroy(): void {
    clearInterval(this.interval);
    this.sub.unsubscribe();
}



createFolder() {
    const addUserRef = this.dialog.open(CreateFolderDialogComponent, {
      width: '250px',
      data: {id: this.id}
    });
    addUserRef.componentInstance.id = this.id;
    addUserRef.afterClosed()
      .subscribe(
        () => setTimeout(
          () => this.retrieveItems(),
          2500))
}
onItemClick(itemId: number) {
    if (this.items.filter(itm => itm.item.itemId == itemId)[0].item.directory) {
      this.router.navigate(['/main/' + itemId]);
    } else {
      this.itemService.downloadItem(itemId)
    }
}

contextMenuAccessControl(item: ItemWithPermission) {
    const accessControlDialogRef = this.dialog.open(AccessControlDialogComponent, {
      width: '80%'
```

```
    });
    accessControlDialogRef.componentInstance.id = item.item.itemId;
  }

  changeSorting(criteria: string) {
    if (criteria == this.sortingParameter) {
      this.isReverseSort *= -1;
    } else {
      this.sortingParameter = criteria;
      this.isReverseSort = 1;
    }
    this.items.sort((a, b) => {
      if (a.item.directory && !b.item.directory) {
        return -1;
      }
      if (!a.item.directory && b.item.directory) {
        return 1;
      }

      switch (this.sortingParameter) {
        case 'name':
          return compare(a.item.name, b.item.name) * this.isReverseSort;
        case 'date':
          return compare(a.item.name, b.item.name) * this.isReverseSort;
        case 'size':
          return compare(a.item.size, b.item.size) * this.isReverseSort;
        default:
          return compare(a.item.name, b.item.name) * this.isReverseSort;

      }
    })

  }
}

function compare(a: number | string, b: number | string) {
  return (a < b ? -1 : 1)
}

@Component({
  selector: 'access-control-dialog',
  templateUrl: './access-control-dialog.component.html',
  styleUrls: ['./main.component.css']
})
export class AccessControlDialogComponent implements OnInit, OnDestroy {
  id: number;
  usersWIP: UserWithItemPermission[];
```

```
interval: any;
me: User;


constructor(public dialogRef: MatDialogRef<AccessControlDialogComponent>,
        public itemService: ItemService,
        public authService: AuthService) {
}

getUsers() {
   console.log('getting users');
   this.itemService.getUsers(this.id).subscribe(data => {
        data.sort((a, b) => {
          if (a.user.email == this.me.email) {
            return -1;
          }
          if (b.user.email == this.me.email) {
            return 1;
          }
          return a.user.email > b.user.email ? -1 : 1
        });
        if (JSON.stringify(data) != JSON.stringify(this.usersWIP)) {
          this.usersWIP = data
        }
      }
   );
}

ngOnInit(): void {
   this.getUsers();
   this.authService.getMe().subscribe((data) => this.me = data);
   this.interval = setInterval(() => {
      this.getUsers()
   }, 2000)
}

setPermission(email: string, permission: string) {
   this.itemService.shareWithUser(this.id, email, permission)
      .subscribe();
}

ngOnDestroy(): void {
   clearInterval(this.interval)   }

removeUser(email: string) {
   this.itemService.removeAccess(this.id, email).subscribe();
}
```

```
}

@Component({
  selector: 'delete-item-dialog',
  templateUrl: './delete-item-dialog.componetn.html'
})
export class DeleteItemDialogComponent {
  @Inject(MAT_DIALOG_DATA) id: number;
  @Inject(MAT_DIALOG_DATA) name: string;
  directory: boolean
  statusCode: number;

  constructor(public dialogRef: MatDialogRef<DeleteItemDialogComponent>, public itemService: ItemService) {
  }

  onNoClick() {
    this.dialogRef.close()
  }

  onSubmit() {
    this.itemService.deleteItem(this.id);
    this.dialogRef.close()
  }
}

@Component({
  selector: 'app-create-folder-dialog',
  templateUrl: './create-folder-dialog.component.html'
})
export class CreateFolderDialogComponent {
  @Inject(MAT_DIALOG_DATA) id: number;
  name: string;
  statusCode: number;
  constructor(public dialogRef: MatDialogRef<CreateFolderDialogComponent>, public itemService: ItemService) {
  }
  onNoClick() {
    this.dialogRef.close()
  }

  onSubmit() {
    this.itemService.createFolder(this.name, this.id).subscribe(
      res => {
        if (res.status == 202) {
          this.dialogRef.close();
        }
        this.statusCode = res.status;
      }
```

```
    )
  }
}

@Component({
  selector: 'app-share-with-dialog',
  templateUrl: 'share-with-dialog.component.html',
  styleUrls: ['./main.component.css']
})
export class ShareWithDialogComponent implements OnInit {
  @Inject(MAT_DIALOG_DATA) id: number;
  email: any;
  groupName: any;
  permission: string;
  private users: User[];
  private groups: GroupWithPermission[];
  myControl = new FormControl();
  myGrControl = new FormControl();
  filteredOptions: Observable<User[]>;
  filteredGroups: Observable<GroupWithPermission[]>;
  statusCode: number;

   constructor(public dialogRef: MatDialogRef<ShareWithDialogComponent>, public itemService: ItemService, public userService: UserService,
public groupService: GroupService) {
  }

  displayFn(user?: User): string | undefined {
    return user ? user.email : undefined;
  }

  displayGr(group?: GroupWithPermission): string | undefined {
    return group ? group.group.name : undefined;
  }

  onNoClick() {
    this.dialogRef.close();
  }

  private _filter(email: string): User[] {
    const filterValue = email.toLowerCase();
    return this.users.filter(option => option.email.toLowerCase().indexOf(filterValue) === 0);
  }

  private _grFilter(name: string): GroupWithPermission[] {
    const filterValue = name.toLowerCase();
    return this.groups.filter(group => group.group.name.toLowerCase().indexOf(filterValue) === 0);
  }
```

```
ngOnInit(): void {
    this.userService.getAll().subscribe(data => {
        this.users = data;
        console.log('1' + data);
        this.filteredOptions = this.myControl.valueChanges
          .pipe(
            startWith(''),
            map(value => typeof value === 'string' ? value : value.email),
            map(name => name ? this._filter(name) : this.users.slice())
          );
      }
    );
    this.groupService.getGroups().subscribe(data => {
      this.groups = data;
      console.log('2' + data)
      this.filteredGroups = this.myGrControl.valueChanges
        .pipe(
          startWith(''),
          map(value => typeof value === "string" ? value : value.name),
          map(name => name ? this._grFilter(name) : this.groups.slice()
          )
        )
    })
}

shareWithUser() {
    this.itemService.shareWithUser(this.id, this.email.email, this.permission).subscribe(data => {
      if (data.status == 200) {
        console.log(this.email + " fff" + this.email.email);
        this.dialogRef.close();
      }
    });
    this.dialogRef.close();
}
shareWithGroup() {
    this.itemService.shareWithGroup(this.id, this.groupName.group.id, this.permission).subscribe(data => {
      if (data.status == 202) {
        this.dialogRef.close();
      }
    });
    this.dialogRef.close();
}
```

ДОДАТОК В

Створення користувацького інтерфейсу для доступу до кафедрального хмарного сховища.

Опис програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ51152_19Б 13-1

Аркушів

Київ 2019

АНОТАЦІЯ

Розділ містить опис частини, яка слугує для роботи з представленням, що є структурною одиницею програмного продукту, та забезпечує виконання головного функціоналу програмного комплексу. Даний модуль відповідає за відображення даних на представлені, та обробку викликів з представлення та подальшу їх логічну обробку.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

У додатку розглядається один з програмних модулів системи — модуль для роботи з відображенням з кодом УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ51152_19Б 12-1, що міститься у файлі main.component.ts. Модуль реалізовано за допомогою програмного директиви @Component. Модуль відповідає за взаємозв’язок представлення та логіки програми.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Призначенням модулю для роботи з представленням є відображення інформації на клієнтській частині, а також обробка дій клієнта, направлення даних введених клієнтом на сервер, або в інший логічний модуль. Це необхідно для того аби додаток був інтерактивним, і є базою будь-якого функціоналу.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Відображати дані та слідкувати за їх оновленням є основною роботою даного модуля. При відкриванні сторінки, модуль звертається на сервер, для того щоб забрати звідти дані. Окрім того модуль регулярно посилає дані на сервер, пби перевірити, чи дані не оновились. І якщо необхідно оновлює дані в представлені.

ТЕХНІЧНІ ЗАСОБИ ЩО ВИКОРИСТОВУЮТЬСЯ

Модуль розроблено з використанням директиви Angular @Component. Окрім того використовуються такі директиви як ngIf, ngFor, ng-template. Цей модуль викликає модуль для з’єднання з API, який в свою чергу використовує HttpClient, для з’єднання з сервером та отримання звідти даних.

ВИКЛИК І ЗАВАНТАЖЕННЯ

Даний модуль автоматично завантажується при завантаженні програми, а отже не потребує ніяких додаткових дій.

Даний модуль реалізовано як набір окремих класів, що викликаються від одного основного.

ВХІДНІ І ВИХІДНІ ДАНІ

Вхідними даними модуля є інформація, яка надходить з сервера.

Вихідними даними модуля можуть виступати дані, які водить клієнт.